Advancing the SystemC Ecosystem

Philipp A Hartmann – Intel Corp.
Jerome Cornet – ST Microelectronics
Martin Schnieringer – Robert Bosch GmbH
Frederic Doucet – Qualcomm, Inc

accellera SYSTEMS INITIATIVE

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Agenda

1. **Accellera Update: What's New and Cooking in SystemC Standardization**
   - Philipp A Hartmann - Intel Corp.
2. **TLM-Serial: Easy and Intuitive Controller Area Network (CAN) Modeling**
   - Jerome Cornet - ST Microelectronics
   - Martin Schnieringer - Robert Bosch GmbH
3. **SystemC for HLS: What Works well for Deployment and Challenges**
   - Frederic Doucet - Qualcomm, Inc.

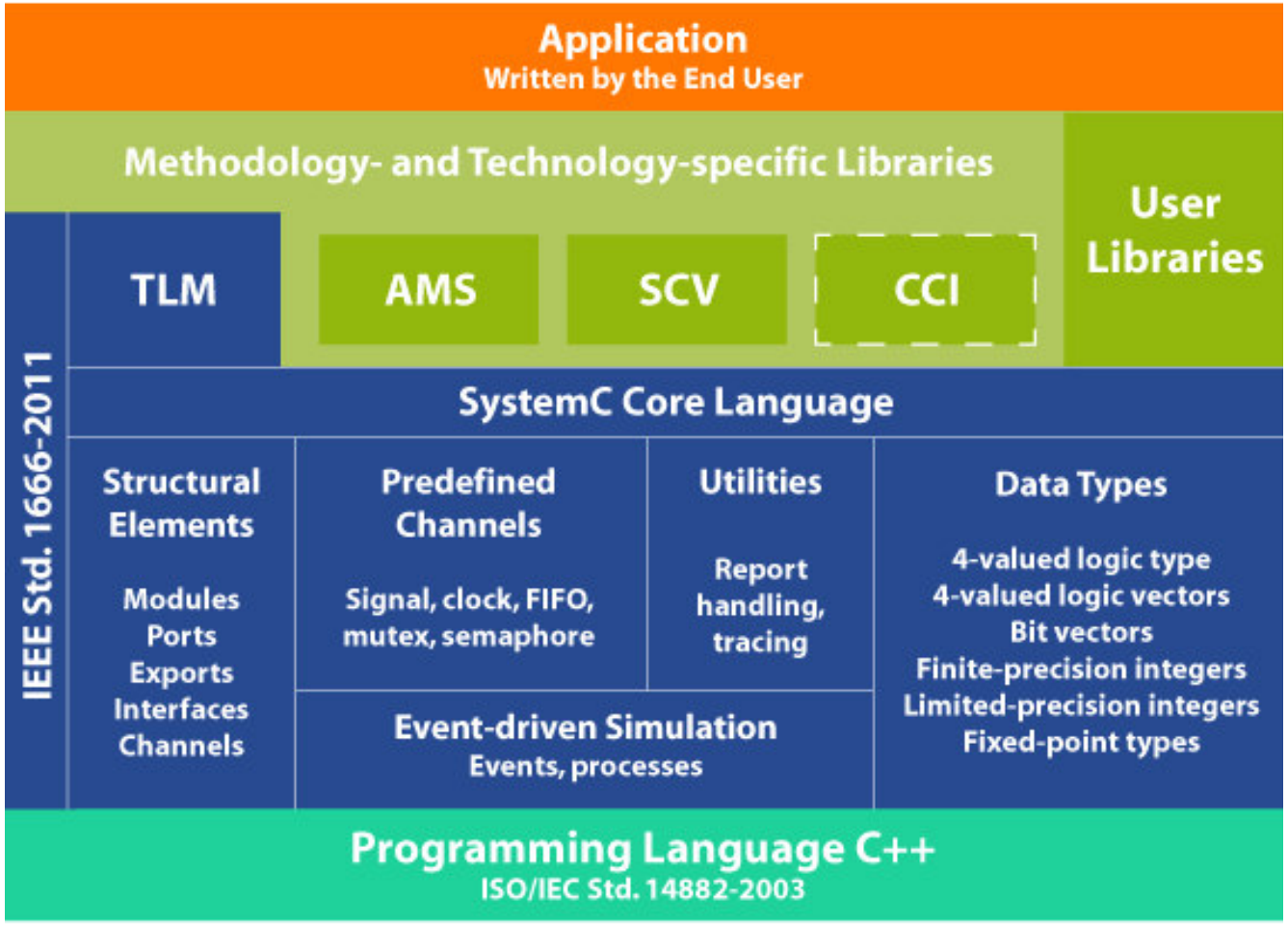
Accellera Update: What's New and Cooking in SystemC Standardization

Philipp A Hartmann, Intel Corp.

accellera SYSTEMS INITIATIVE

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

SystemC Overview



- C++-based language, widely used for
  - system-level modeling, design and verification
  - architectural exploration, performance modeling
  - analog/mixed signal modeling
  - software development
  - high-level synthesis
- Defined by Accellera WGs, ratified as IEEE Std. 1666™-2011

accellera SYSTEMS INITIATIVE

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

SystemC Language Working Group

- **Charter:** Responsible for the **definition** and **development** of the **SystemC core language**, the foundation on which all other SystemC libraries and functionality are built.
- **Current status**
  - SystemC/TLM 2.3.2 release at DVCon Europe
  - Currently collecting, addressing, refining proposals and errata towards IEEE std. update
  - Adding extensions to the core language (e.g. as needed by other SystemC-related WGs)
- **Plans for 2017+**
  - Reconvene IEEE P1666 WG for update of IEEE Std. 1666™
  - Improvements to SystemC Datatypes as needed for HLS and beyond (dedicated sub-WG has been formed, **contributors needed!**)

SystemC 2.3.2 release

- **SystemC 2.3.2** release available now!
  - Maintenance release with some new features
  - Licensed under Apache 2.0 License
  - http://accellera.org/downloads/standards/systemc
- Updated compiler and platform support
  - Support for latest compiler versions (GCC 7.0, Clang 5.0, MSVC 2017)
  - Support for AArch64 platform
  - Windows DLL support
  - Experimental CMake build system
- Tons of bug fixes and cleanups

SystemC 2.3.2 – release highlights

- Foundation for C++11/14/17 enablement
  - Based on DVCon Europe 2016 proposal - *Moving SystemC to a New C++ Standard*
  - Removes embedded Boost dependency on modern platforms
  - Override default selection by defining SC_CPLUSPLUS during library/model build (needs to be consistent – see documentation)
- Centralized global name registry to enable CCI naming requirements
  - `bool sc_register_hierarchical_name(const char* hierarchical_name);`
  - `const char* sc_get_hierarchical_name(const char* hierarchical_name);` (persistent name pointer)
- Improved **`sc_time`** conversions from/to strings

SystemC 2.3.2 – release highlights (cont.d)

- Querying active event notifications
  - New **`triggered()`** function returns `true`, if an event has been notified during the previous delta (or immediately in the current evaluation phase)

```
wait( event0 | event1 );
if( event0.triggered() ) /* event0 actions */;
if( event1.triggered() ) /* could also be true! */;
```

- VCD tracing enhancements
  - Tracing of **`sc_time`**, **`sc_event`**
  - Accuracy fixes for delta cycle tracing
  - Support for VCD hierarchical scopes

SystemC TLM Working Group

- **Charter:** The Transaction-level Modeling Working Group (TLMWG) is responsible for the definition and development of methodology and add-on libraries for **transaction-level modeling** in SystemC.
- **Current status**
  - Accellera TLM-2.0 is part of IEEE 1666-2011
  - PoC implementation 2.0.4 bundled with SystemC 2.3.2
- **Plans for 2017+**
  - Work on TLM interfaces, extensions, and guidelines to improve modeling of protocols beyond memory-mapped I/O
  - „TLM signals“; serial, bi-directional communication, ...

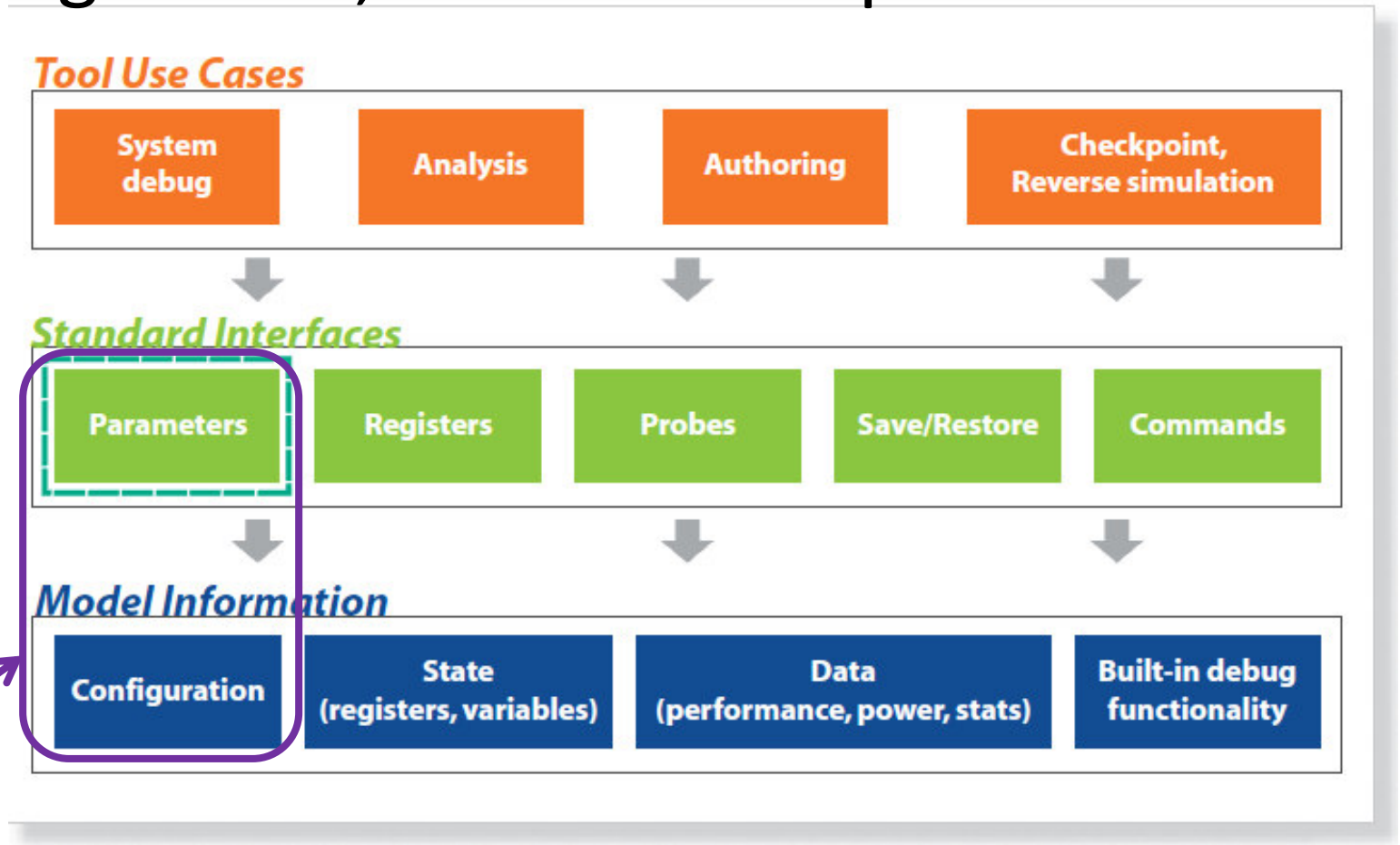
SystemC TLM 2.0.4 changes

- Add new socket base class **`tlm_socket_base_if`**, enabling non-templated access to
  - protocol types (RTTI)
  - base (ex)ports
  - socket category (`TLM_INITIATOR_SOCKET`, `TLM_TARGET_SOCKET`, …)
  - bus width
- Improved error handling in convenience sockets
  - Errors now consistently include their origin (affected socket name/kind)
- Move parts of TLM-2.0 library into prebuilt SystemC (shared) library

SystemC Synthesis WG

- **Charter:** To define the SystemC **synthesis subset** to allow synthesis of digital hardware from high-level specifications.
- **Current status**
  - Working on second version of the SystemC Synthesis Subset standard
- **Plans for 2017+**
  - Update and finalize support C++ 2011/2014 and advanced datatypes support
  - Gather and evaluate additional requirements
    (for example unions, `std::array`, attributes etc.)

Configuration, Control & Inspection WG



Goal: Standardizing interfaces between models and tools

CCI WG status

- Configuration draft standard is ready for public review
  - Tutorial
    - Previewed at DVCon EU (Oct ‘16), DVCon US (Feb ‘17), DVCon India (Sep ‘17)
  - Language Reference Manual
  - Proof-of-concept implementation
  - 20+ examples
- Public review period will be 90 days
- Community feedback is highly valued

SystemC Analog/Mixed-Signal WG

- **Charter:** The SystemC AMSWG is responsible for the standardization of the SystemC AMS extensions, defining and developing the language, methodology and class libraries for **analog, mixed-signal and RF modeling** in SystemC.
- **Current status**
  - Feature development for SystemC AMS 2.1
  - Proposal for generalized small-signal analysis (AC) under discussion
- **Plans for 2017+**
  - Review and Update of SystemC AMS User's Guide

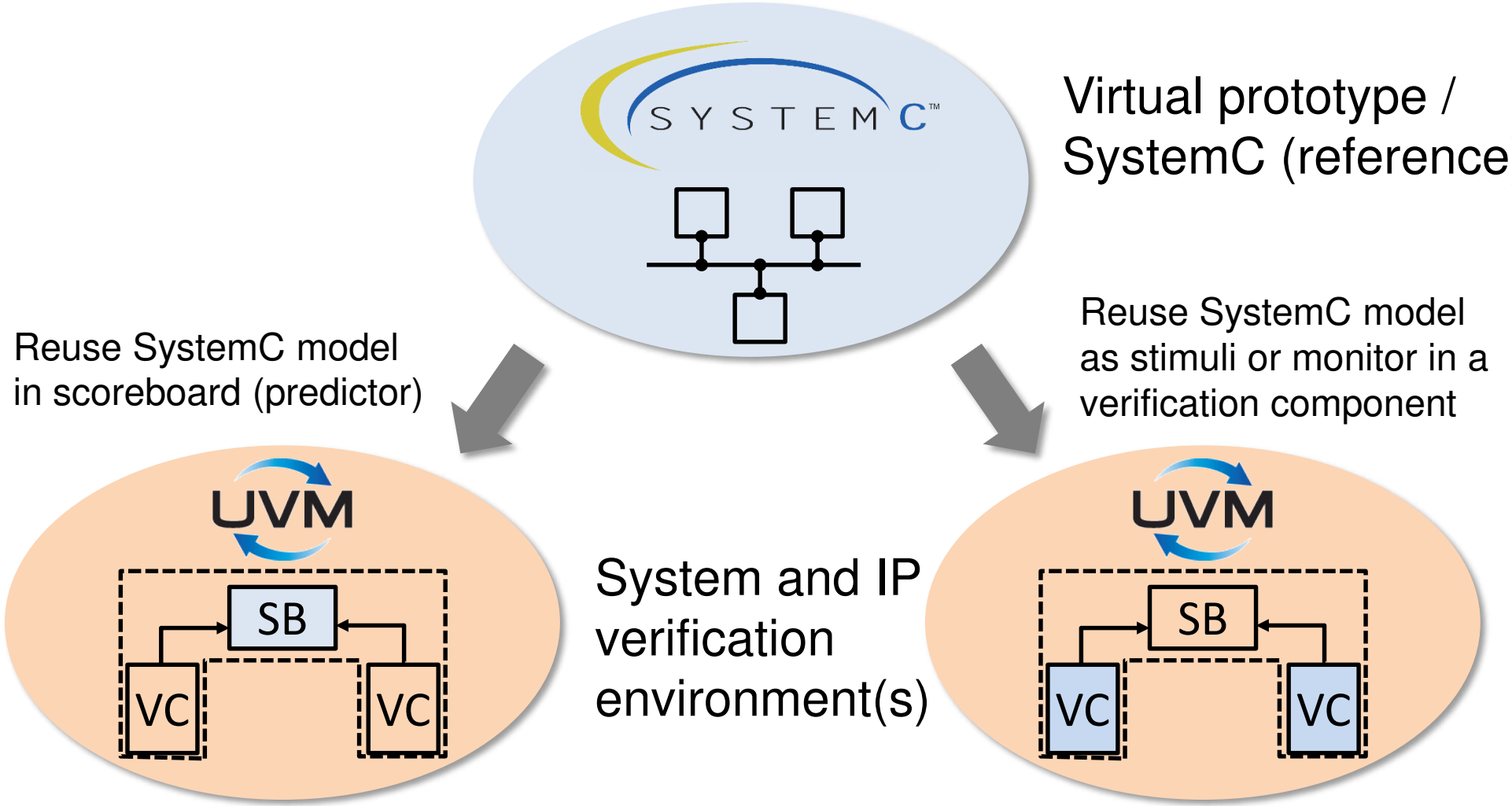
SystemC Verification WG

- **Charter:** The VWG is responsible for defining **verification extensions** to the SystemC language standard, and to enrich the SystemC reference implementation by offering an add-on libraries to ease the deployment of a verification methodology based on SystemC.
- **Current Status**
  - Register API (backdoor) added
  - UVM 1.2 reporting API added
  - SystemC 2.3.2 compatibility
  - Stability review
- **Plans for 2017+**
  - Release shortly after DVCon Europe 2017
  - Smart Pointer implementation
  - IEEE 1800.2-2017 compatibility
  - Register API (frontdoor/backdoor) completed

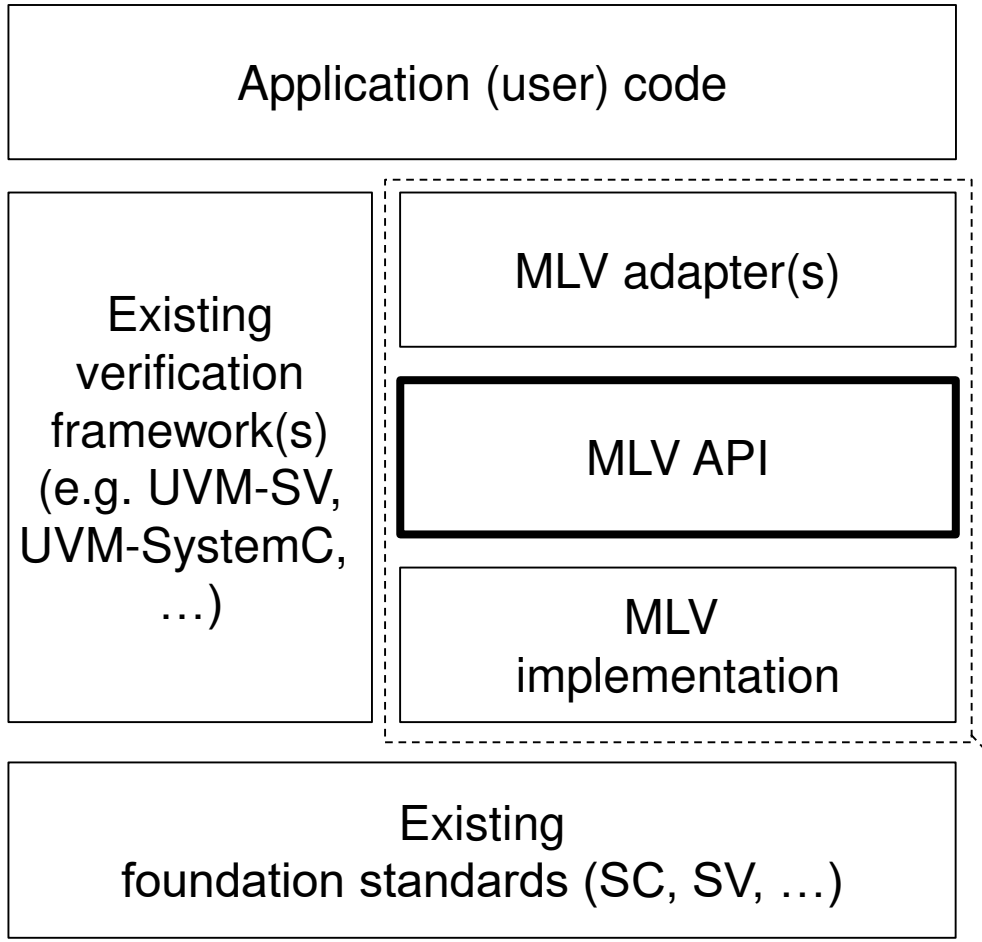
Multi-Language Verification WG

- **Charter:** The mission of the MLVWG is to create a standard and functional reference for interoperability of multi-language verification environments and components.
- **Current Status**
  - MLV requirements and use cases have been identified and documented
  - MLV API definition ongoing for communication (TLM), phasing and configuration
  - Development of a demonstrator based on contributions (UVM-ML, UVM-SystemC)
- **Plans for 2017+**
  - Finalize first demonstrator as starting point for PoC development
  - Alignment with SystemC standard to differentiate design (model) elaboration from test bench elaboration
  - Stabilize MLV API for documentation and LRM development
- **Your participation is highly appreciated!**

Seamless SystemC model reuse in verification



Multi-Language Verification Architecture



(Multi-language) verification environment developed by end-user

Multi-language framework adapters developed by EDA and/or Framework provider(s)

Standardized abstract interface definitions (“Accellera standard”)

Implementation offering functionality like backplane, phasing, communication, etc.

MLV Proof-of-concept

Advancing Standards Together

- Share your experiences
  - Visit www.accellera.org and join community forums at forums.accellera.org
- Show your support
  - Record your adoption of standards
- Become an Accellera member
  - Join working groups
- **SystemC Evolution Day** on Wednesday
  - 08:30 – 17:30, TU Munich (Arcisstraße 21)
  - Current and future needs for SystemC/TLM/HLS, … and your **favorite topics**!
  - http://accellera.org/news/events/systemc-evolution-day-2017
  - For last-minute registration, please contact me (Philipp Hartmann, Intel)

Questions?

SystemC Evolution Day on Wednesday
October 18th, 08:30-17:00, TU Munich

http://accellera.org/news/events/systemc-evolution-day-2017

accellera SYSTEMS INITIATIVE

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

TLM-Serial: Easy and intuitive Controller Area Network (CAN) modeling

Jerome Cornet - ST

Martin Schnieringer – Robert Bosch GmbH

BOSCH

accellera SYSTEMS INITIATIVE

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Agenda

- Introduction
- What is CAN?
- Modeling a Controller Area Network (CAN)
  - What is modeled at all?
  - CAN payloads & sockets
  - Creating a CAN node
  - Connecting the CAN nodes using a „bus“
  - Status
- Summary and outlook

Introduction

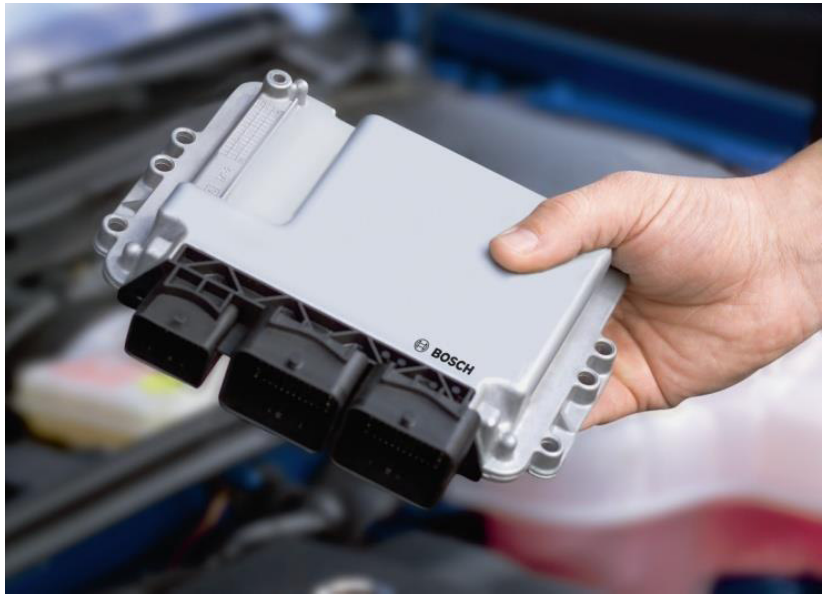
Motivation:

- Early & efficient SW development on Virtual Prototypes of Electronic Control Units (ECU)
- ECU (network) is assembled from IP of different vendors communicating via serial protocols e.g. CAN-FD

Avoid effort when connecting simulation models

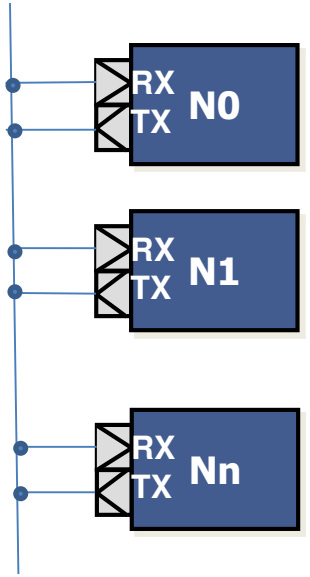
Goal:

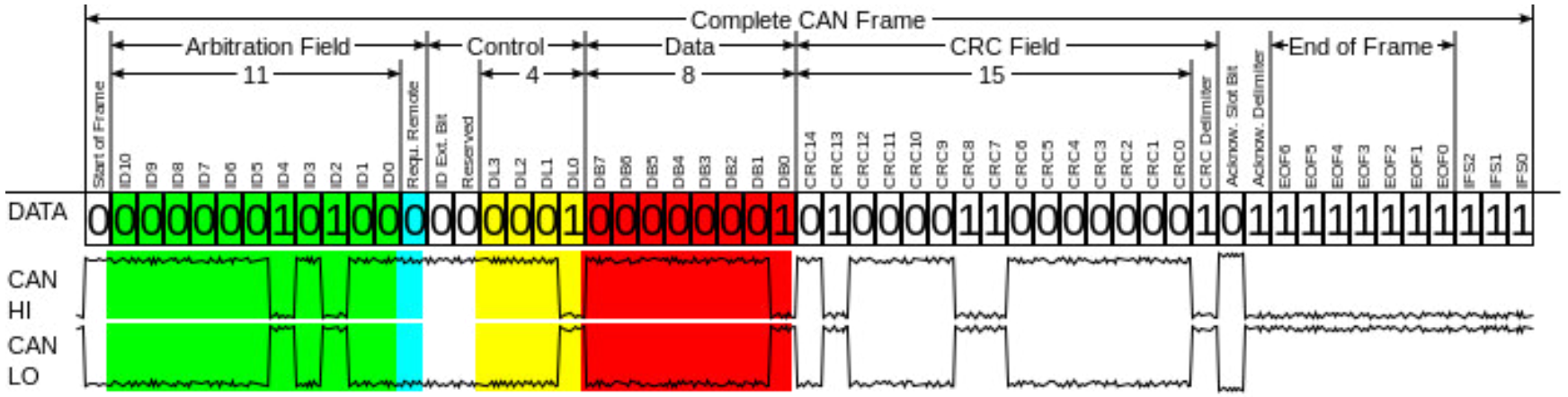
- Establish SystemC TLM modeling standard for serial protocols, initial focus on CAN



What is CAN?

- Serial, frame based field bus, half duplex
- Multi-master with broadcasting
- Message based, receiver filters CAN frames
- Arbitrating





Source: wikipedia.org

What is modeled at all?

- Controller Area Network with Flexible Data-Rate (CAN-FD) according to ISO11898-1 second edition 2015-12-15
- CAN(-FD) protocol on Transaction Level (TL), timing accurate on CAN frame boundaries
  - Timing based on worst case bit stuffing
  - RTL co-simulation support
- Full arbitration
- Error injection e.g. CRC, BIT
- CAN „bus“ that connects the nodes

CAN payloads

- Three types of CAN payloads: For data, error and overload frames
- All payloads inherit from **`can_payload_base`**
  - Holds nominal bit time with accessors, common to all frames

`can_payload`: Used for (extended) CAN/CAN-FD data, remote frames

`can_error_payload`: Used to broad-cast errors to nodes

`can_overload_payload`: Used to prolong bus busy time

can_payload

- Stores CAN data, modes and phases, offers API to all members

```
bool     m_remote;       /// True when the frame is a remote frame
bool     m_extended;     /// True when the identifier is extended
bool     m_fd_format;    /// True if the frame is CAN-FD
uint32_t m_identifier;   /// The CAN ID
phase_t  m_phase;        /// Phase of the frame
sc_time  m_data_bit_time; ///< Data Bit Time of a CAN-FD frame.
bool     m_ack;
bool     m_brs;          /// Bit Rate Switch for CAN-FD
bool     m_error_passive; //Used to indicate whether node is
                                 error passive or active.
```

The CAN socket

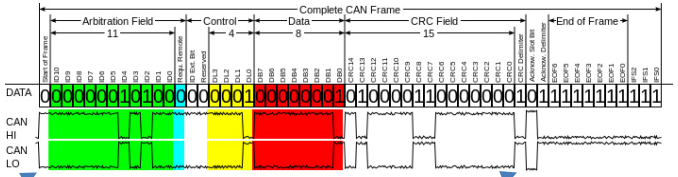
- **`can_socket`** has RX/TX path
- Offers non-blocking and blocking transmit API
- Is responsible for arbitration and forwarding interface calls to the CAN node when applicable
- Implements binding

```
can_tx_status_t transmit_b(can_payload & payload);
can_tx_status_t transmit_nb(can_payload & payload);
void transmit_error(can_error_payload & error_payload);
void transmit_overload(can_overload_payload & overload_payload);
void cancel(can_payload & payload);

bool can_bus_available() const;
void set_socket_id(int socket_id);
int get_socket_id();
```

Creating a CAN node

- Node inherits from **`tlm_serial_if`**
- Implements the interface functions




```
void can_node_simple::transmit_core(tlm_serial::can_payload & payload, int socket_index)
{
  if(payload.get_phase() == tlm_serial::CAN_SOF_AND_ARBITRATION) {
    :
  } else if(payload.get_phase() == tlm_serial::CAN_ACK) {//Pick-up data here
        :
}
void can_node_simple::transmit_error_core(tlm_serial::can_error_payload & error_payload,
int socket_index) {
  return;
}
void can_node_simple::cancel(tlm_serial::can_payload & payload, int socket_index) {
  SC_LOG(name(), "cancel" << ": Ignore latest received payload.");
  return;
}
```

Creating a CAN node

- Example using the blocking transmit call

```
//Populate CAN payload upfront
tlm_serial::can_tx_status_t l_tx_status = can_socket.transmit_b(m_tx_payload);
switch(l_tx_status) {
      case tlm_serial::TX_OK:
        SC_LOG(name(), "thrd_main_action" << ": Won CAN bus arbitration, CAN frame sent :)");
        break;
      case tlm_serial::TX_ERROR:
        SC_LOG(name(), "thrd_main_action" << ": An ACK, BIT, CRC, FORM or STUFF error occured.");
        break;
      case tlm_serial::TX_ARBITRATION_LOST:
       SC_LOG(name(), "thrd_main_action" << ": Lost CAN bus arbitration, will retry ...");
        break;
      case tlm_serial::TX_CANCELLED:
        SC_LOG(name(), "thrd_main_action" << ": CAN frame got cancelled e.g. via reset.");
        break;
```

Connecting the CAN nodes using a „bus“

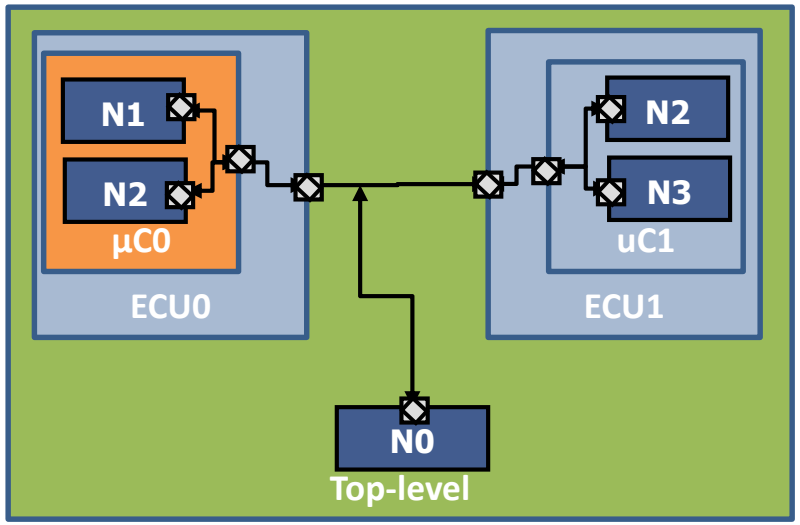
- The can_bus binds all sockets accross the hierarchy levels
- One „bus“ instance is used per hierarchy

µC-level

```
tlm_serial::can_bus BUS_µC0;
N1.can_socket(BUS_µC0);
N2.can_socket(BUS_µC0);
BUS_µC0(can_socket);
```

Top-level

```
tlm_serial::can_bus BUS_TL;
ECU0.can_socket(BUS_TL);
NODE0.can_socket(BUS_TL);
ECU1.can_socket(BUS_TL);
```



Timing points (phases)

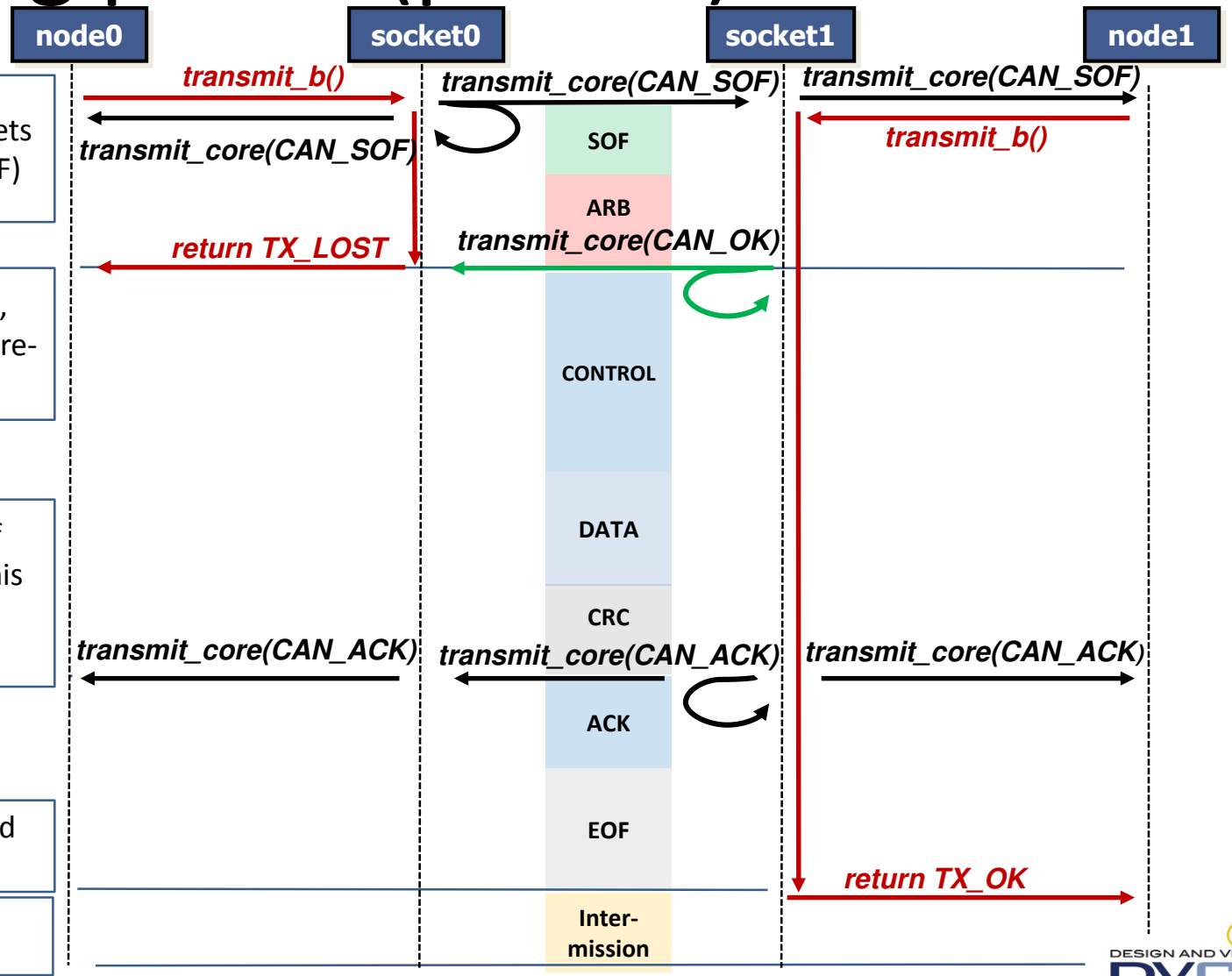
Initiate TXN, SOF communicated to all sockets (payload of first transmit_b() is used for SOF)

End of arbitration field, losing node returns, broadcast of winning node to all sockets -> re-calculates EOF time in all sockets.

ACK is communicated to node at the end of the ACK slot (first bit of the ACK field). At this point all data is available e.g. data could be late due to sending RTL node.

Winning node returns from blocking call and socket notifies ***bus idle event***.

bus idle event fires after interframe space bits have been sent



Status

- Work in progress, limited documentation and rules
- All defined CAN scenarios as uploaded to accellera are supported
- Code and examples on private github repository
  - Tested with SystemC 2.3.1 using gcc and MSVC12
  - Simple batch runs for regression tests
  - Release to Accellera after legal clearance

Summary and outlook

Summary

- First version of CAN(-FD) SystemC TLM implementation available
- Simple and easy to use due to CAN socket convenience layer
- Powerful and flexible. Covers all relevant scenarios, imposes no modeling limitations
  - RTL co-simulation support

Outlook

- Address further serial protocols e.g. SPI, I2C
- More testing on
  - RTL co-sim e.g. with SystemC RTL node
  - Non-blocking CAN socket API

Questions

References

- ISO11898-1 second edition 2015-12-15 - CAN with Flexible Data-Rate (CAN-FD) https://www.iso.org/standard/63648.html
- Serial TLM requirements and scenarios (from Bosch, Infineon, ST etc.) https://workspace.accellera.org/apps/org/workgroup/tlmwg/download.php/14231/SerialTLM_requirements_and_scenarios.doc

SystemC for High-level Synthesis:
what works well for deployment and challenges

Frederic Doucet

Qualcomm Technologies Inc.

San Jose, CA, USA

accellera SYSTEMS INITIATIVE

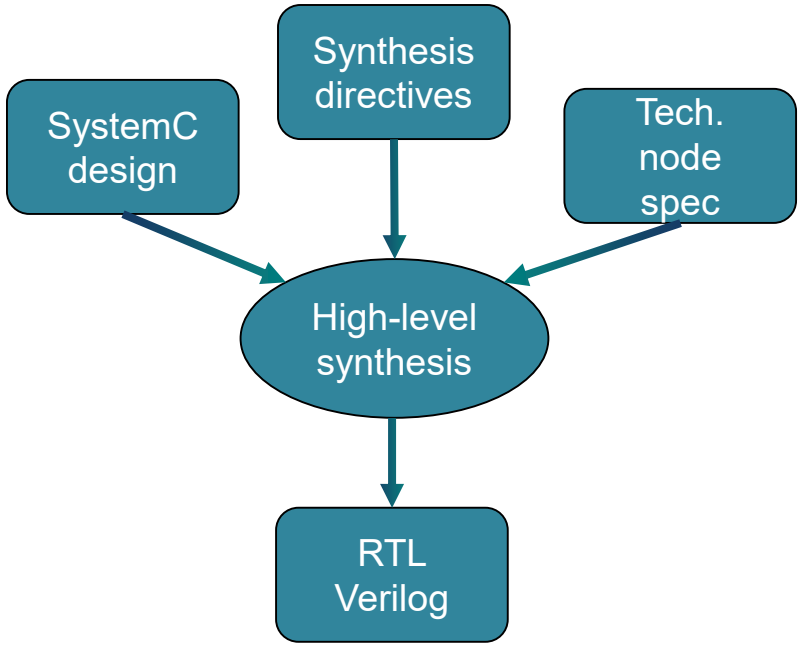
2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Outline

- Introduction to High-level synthesis
- Synthesizing SystemC: how it works
- What is abstracted out in SystemC and refined by HLS
- The pitfalls of informal abstraction
- Examples of experienced hardware designers falling in the pitfalls
- Going forward with HLS

What is High-level Synthesis

HLS tool transforms synthesizable SystemC code into RTL Verilog



The HLS engine

- … precisely characterizes delay / area of all operations in a design
- … schedules all the operation over the available clock cycles
- … can optionally increase latency
  - to get positive slack and
  - to share resources and reduce area
- … generates RTL that is equivalent to input SystemC
  - Pipe depths / latencies decided by HLS tool scheduler

Overview of HLS Usage in Industry

What gets designed:

- Large datapaths, control mixed with datapath, etc., .. thousands of tapeouts!!

Benefits:

1. Fast design turnaround:
   - Quickly implement large (or micro-architecture) changes and regenerate RTL
   - Allows for fast micro-architecture exploration and design optimizations
2. High-level verification : huge productivity benefits at SystemC level
   - Bit match datapath “compute()” functions,
   - Generate and analyze code coverage on “higher abstraction” SystemC code
   - Bugs are mostly in integration with other non-HLS RTL blocks

Overview of HLS Usage in Industry

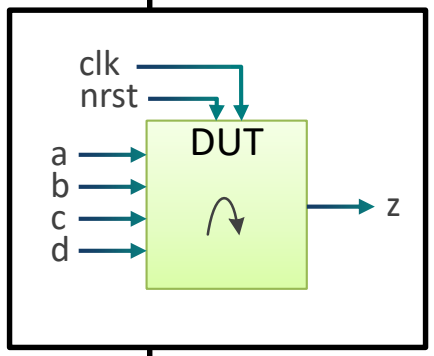
- HLS used on variety of designs, from very small to very large!
- Example of sizes of synthesized SC_CTHREAD
  - Small ~1k - 10k instances
  - Large ~100k instances
  - Very large ~500k instances
- It is very important to find proper abstraction for very large blocks
  - Good hardware designer knows what RTL structure to expect

    How to capture in SystemC and have HLS generate the desired result
  - Fast HLS run-time usually correlates with good Quality-of-Results (QOR)
  - Timing characterization is also very important, it has significant impact on instance counts and area (pipeline depths, upsized operators etc)

Hardware Design with SystemC and HLS

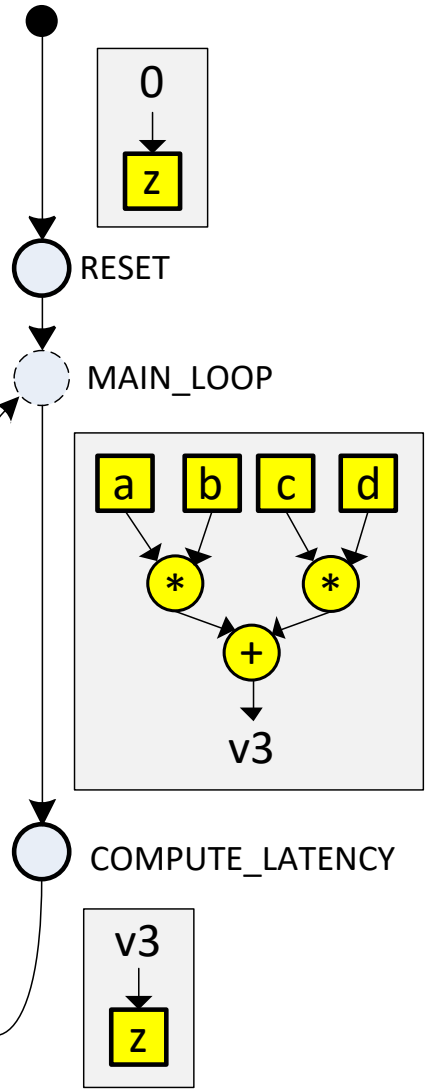
- SystemC: syntax for hardware modeling framework in C++
  - Modules
  - Ports
  - Connections
  - Processes
- Inside a process is C++ code describing the functionality
  - DSP processing
  - Control logic
  - Etc.

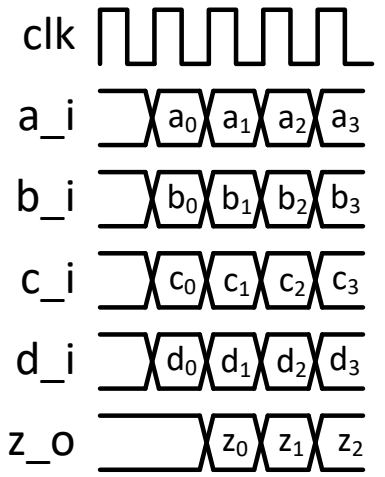
```
15 SC_MODULE(DUT)
16 {
17     sc_in <bool>     clk;
18     sc_in <bool>     nrst;
19     sc_in <int>      a;
20     sc_in <int>      b;
21     sc_in <int>      c;
22     sc_in <int>      d;
23     sc_out<int>      z;
24
25     SC_CTOR(DUT) {
26         SC_THREAD(proc);
27         sensitive << clk.pos();
28         reset_signal_is(nrst, false);
29     }
30
31     void proc() {
32         z = 0;
33         RESET:
34         wait();
35         MAIN_LOOP:
36         while (true) {
37             int v1 = a * b;
38             int v2 = c * d;
39             int v3 = v1 + v2;
40             COMPUTE_LATENCY:
41             wait();
42             z = v3;
43         }
44     }
45 };
```



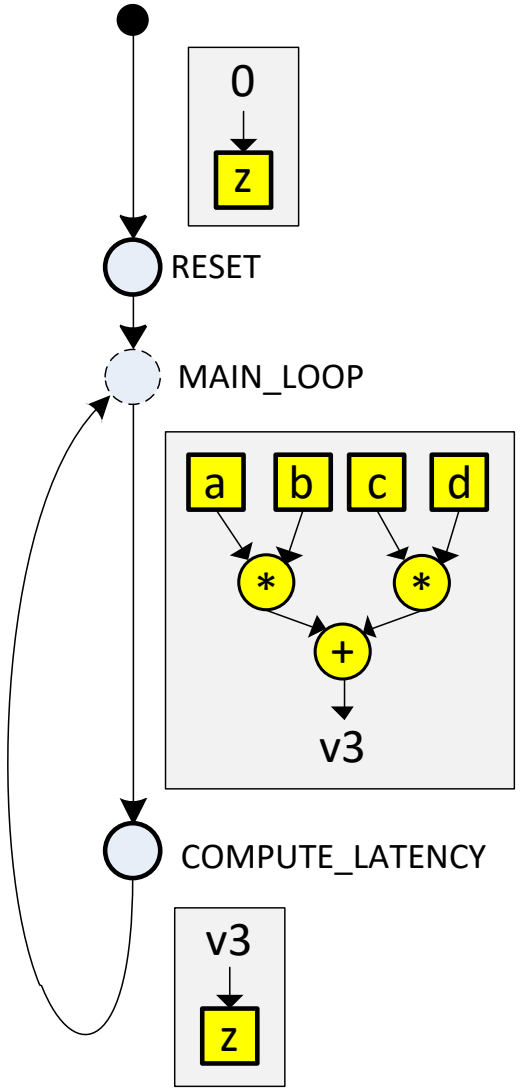
Behavior of Synthesizable SystemC Process

```
15 SC_MODULE(DUT)
16 {
17     sc_in <bool>    clk;
18     sc_in <bool>    nrst;
19     sc_in <int>     a;
20     sc_in <int>     b;
21     sc_in <int>     c;
22     sc_in <int>     d;
23     sc_out<int>     z;
24
25     SC_CTOR(DUT) {
26         SC_THREAD(proc);
27         sensitive << clk.pos();
28         reset_signal_is(nrst, false);
29     }
30
31     void proc() {
32         z = 0;
33         RESET:
34         wait();
35         MAIN_LOOP:
36         while (true) {
37             int v1 = a * b;
38             int v2 = c * d;
39             int v3 = v1 + v2;
40             COMPUTE_LATENCY:
41             wait();
42             z = v3;
43         }
44     }
45 };
```





Simple HLS : Cycle-accurate design



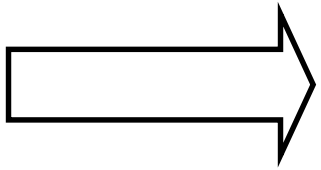
Clock period: 5ns

op delays for technode:

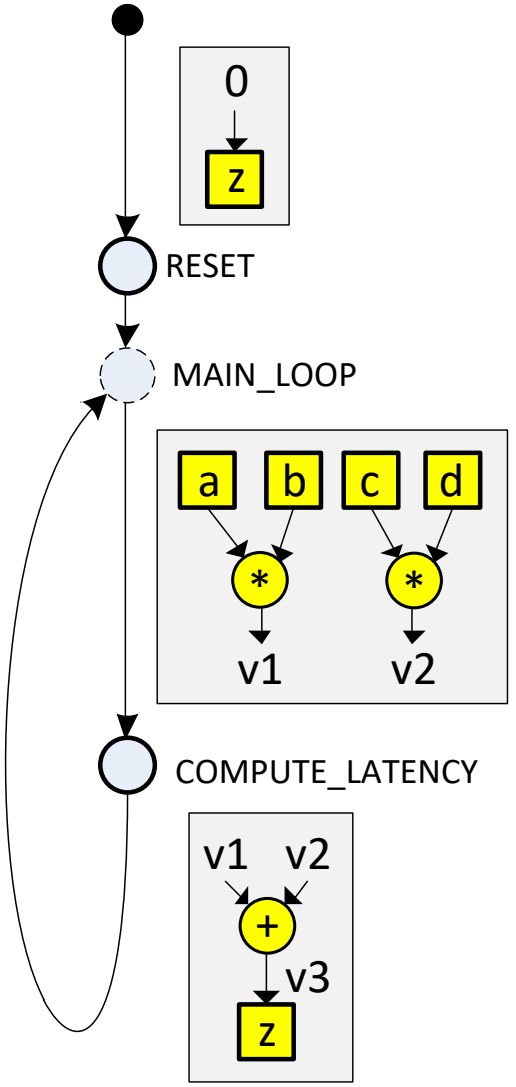
- mul: 4ns
- add: 2ns

Synthesis directive:

Cycle accurate design

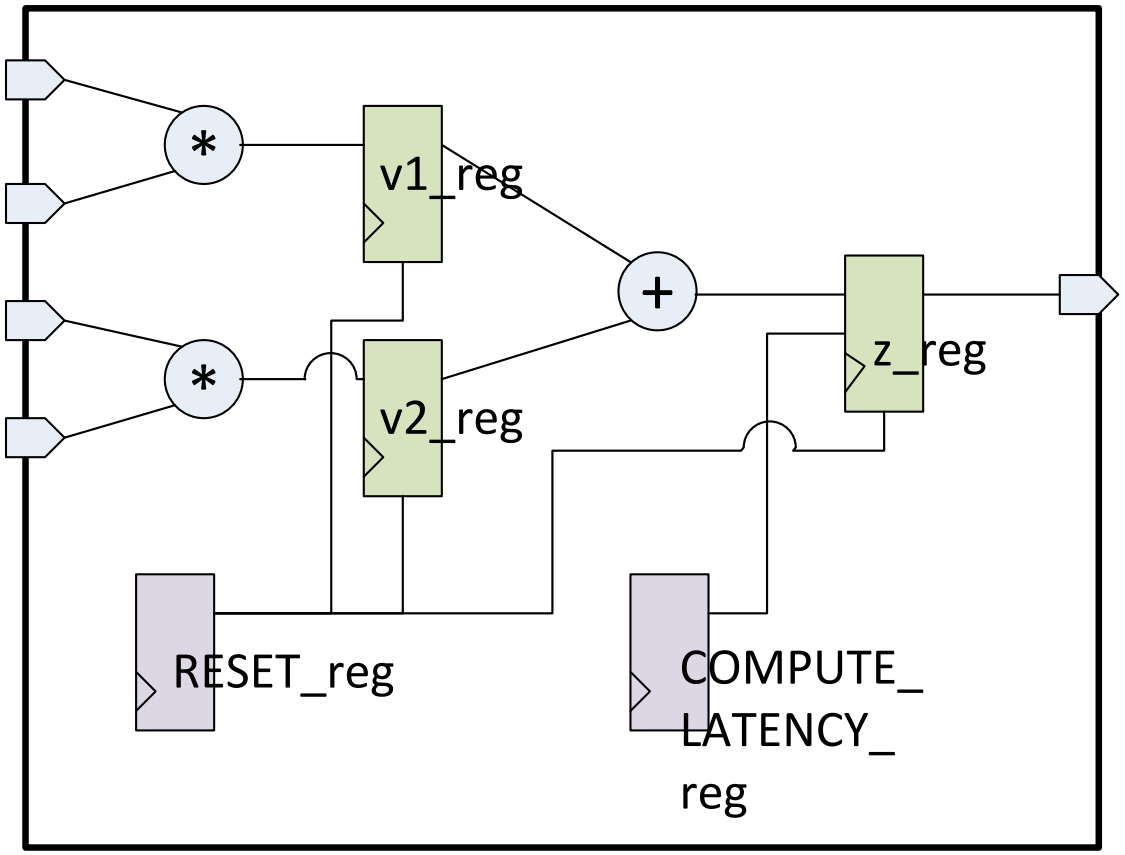


To get positive slack, the '+' operation is moved across the COMPUTE_LATENCY wait()

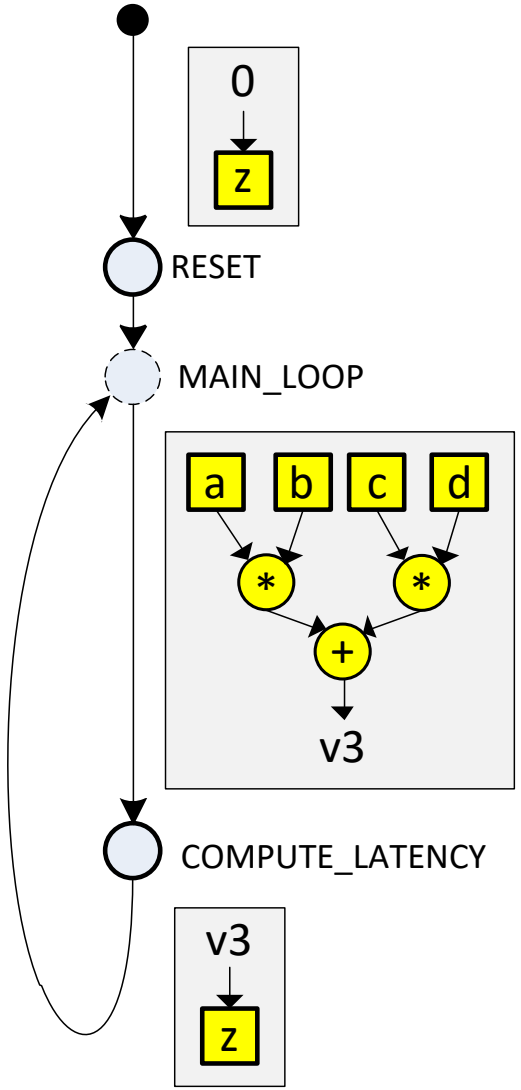


What the HLS tool do

1. Allocate arithmetic/logic resources
2. Map operations to resources
3. Allocate registers to store data in-flight
4. Connect datapath I/O and components
5. For all states, allocate the control registers
6. Generate current / next state logic and
7. Connect control signals to datapath register / mux enables



HLS Directive: minimize resources

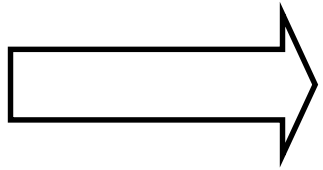


Clock period: 5ns

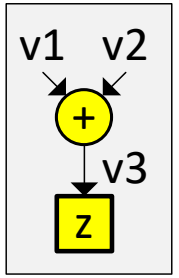
op delays for technode:

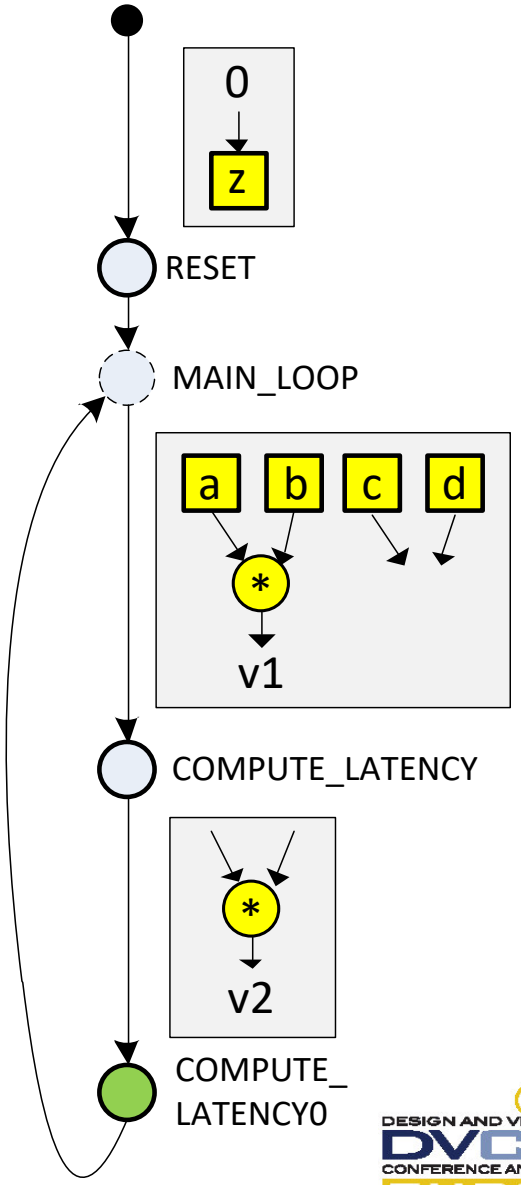
- mul: 4ns
- add: 2ns

Synthesis directive:
Minimize resources



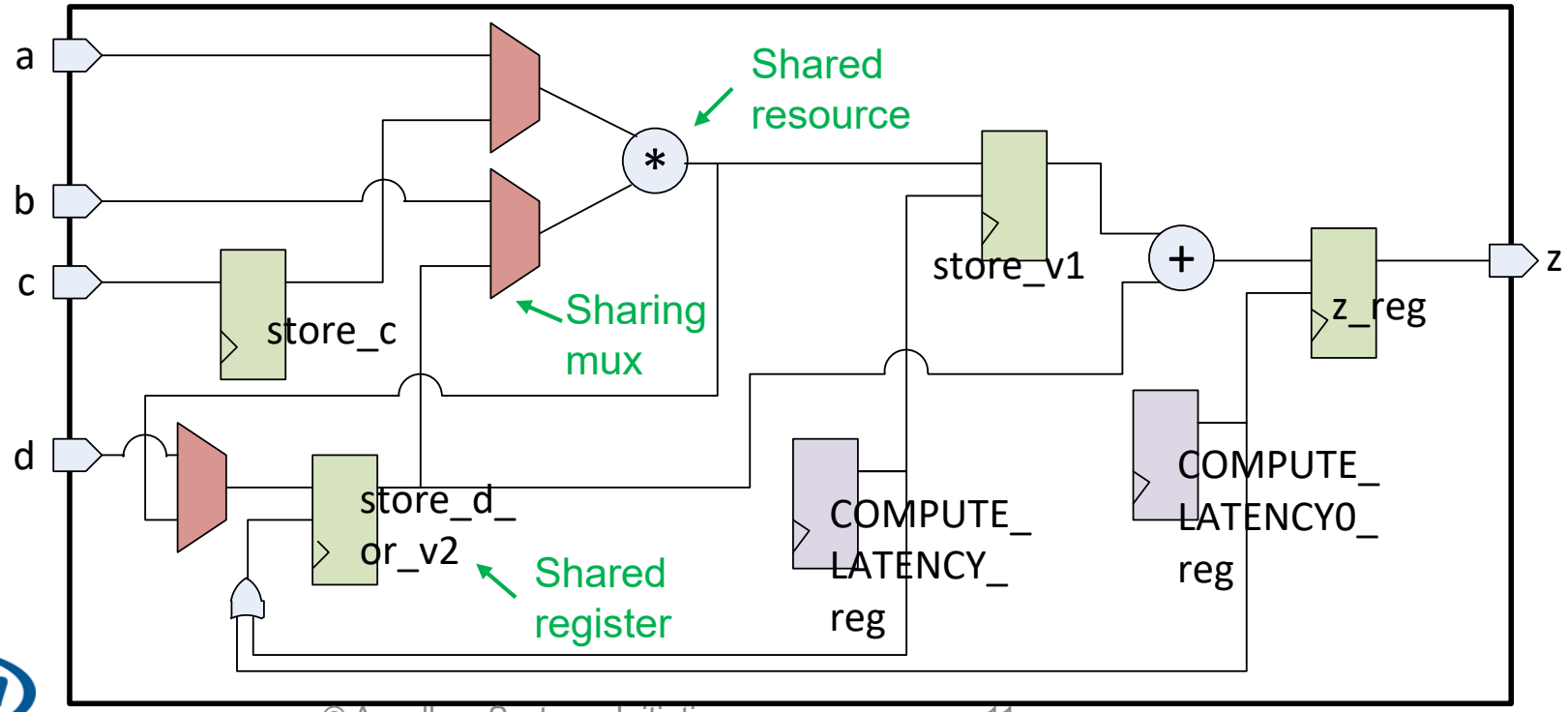
Latency is increase to use only one multiplier for two multiplications





Very different micro-architecture..

- One multiplier now shared for two multiplications
- Sharing muxes and registers have been added around the multiplier
- FSM changed driving new enables for sharing muxes and registers



What is abstracted out in SystemC? (and refined by the HLS tool?)

Abstract in SystemC, Refined by HLS

1. Operations to resource bindings and sharing muxs
   - Resource sharing depends on the synthesis directives (performance or area?)
2. Internal registers
   - Values in flight need to be registered
   - Depends on how operation are mapped to resources, which depends on the synth directives
3. FSM states and transitions:
   - wait() statements are converted to FSM states (in code, and added by tool)
   - transitions between waits are FSM transitions
   - current / next state logic generated by the tool

Abstraction for Hardware Designer

- Hardware designers usually
  1. clearly understands the resource sharing, register and FSM abstractions
     - explicitly written in RTL designs, making it tedious and error, not easily changeable
  2. think structurally
     - “can you draw the block diagram on the board”
     - “what do you expect to see in the RTL?”
  3. are not C++ experts
     - Keep the intro example simple, and relate them to their RTL experience…
  4. have low expectations on EDA tool

- HLS is usually an easy sell … unless…

Selling Informal “High-Abstraction”

- For some algorithm person:
  - *“I like to write high-level algorithm code in C++ “*
  - *“Verilog has way too much details!”*
- For some management / DV :
  - Less lines of code! Easier to understand and debug!
  - Faster simulation speed!
- For some software person:
  - TLM channels instead of signal/toggling! Encapsulation!
  - Vectors and iterators, templates and other advanced C++!

Problems with Informal “High-Abstraction“

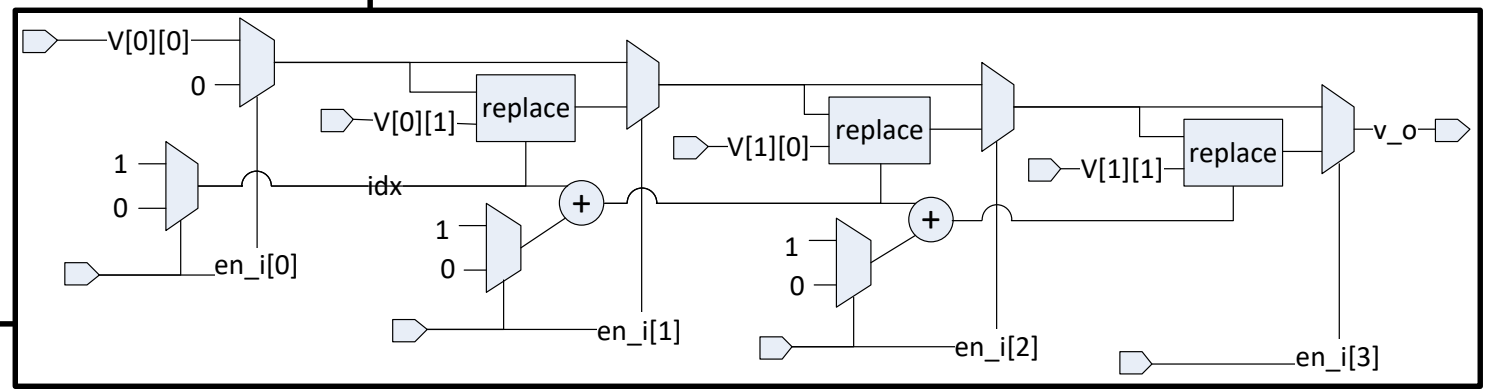
- It can create unreasonable expectations of what the HLS tool will do
  - *“Tool will understand my coding style, it is so clean, it is obvious!"*
- The consequence is designs with poor QOR
  - area too large, too much congestion, etc.
- Designer will spend significant time to re-coding to hit the QOR targets
  - This makes design management *very* nervous about HLS
- Such experience in design groups foster negative sentiment for further HLS
  - And these are hard to reverse…
  - *“HLS? You like bloated designs? No HLS on my projects!”*

Even if abstraction well understood, first HLS project is rarely smooth: need new flow, training, understand the tools, hands-on support, etc.

Experienced Designers Stumbling with Muxes (1/2)

```
SC_MODULE(DUT)
{
  typedef sc_uint<NROWS*NCOLS>            en_t;
  typedef sc_biguint<NROWS*NCOLS*NBITS>   out_t;
  ...
  sc_in <sc_uint<NBITS> >   v_i [NROWS][NCOLS];
  sc_in <en_t>              en_i; // one bit per element
  sc_out<out_t>             v_o;
  ...
  void proc() {
      ...
      int     idx = 0;
      out_t   v   = 0;
      en_t    en  = en_i.read();
      for (int row=0; row<NROWS; row++) {   // unroll
        for (int col=0; col<NCOLS; col++) { // unroll
          if (en[row*NCOLS+col]) {
            const int lb = idx*NBITS;
            v.range(lb+NBITS-1,lb) = v_i[row][col];
            idx++;
          }
        }
      }
      v_o = v;
      ...
  }
};
```

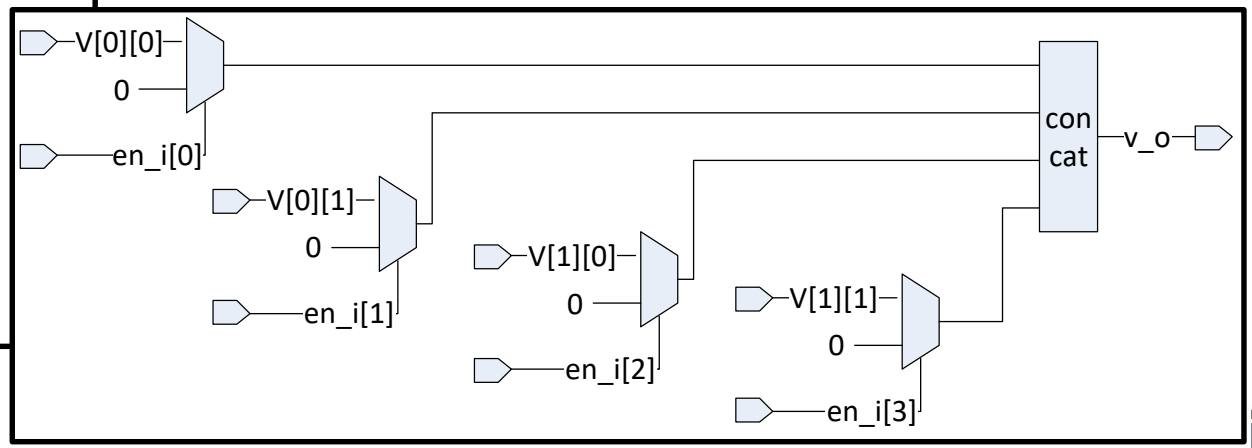
- Code to serialize (and later deserialize) a data matrix
- An N * M array element gets copied to a bit vector if the enable is true
- *The index for the bit vector is in the enable condition…*
  - *Creates a chain of full size replace muxes (on out_t) selected with adders outputs*



Experienced Designers Stumbling with Muxes (2/2)

```
SC_MODULE(DUT) {
  typedef sc_uint<NROWS*NCOLS>            en_t;
  typedef sc_biguint<NROWS*NCOLS*NBITS>   out_t;
  ...
  sc_in <sc_uint<NBITS> >   v_i [NROWS][NCOLS];
  sc_in <en_t >             en_i;
  sc_out<out_t>             v_o;
  ...
  void proc() {
      ...
      int     idx = 0;
      out_t   v   = 0;
      en_t    en  = en_i.read();
      for (int row=0; row<NROWS; row++) { // unroll
        for (int col=0; col<NCOLS; col++) { // unroll
          if (en[row*NCOLS+col]) {
            const int lb = idx*NBITS;
            v.range(lb+NBITS-1,lb) = v_i[row][col];
            idx++;
          }
          idx++;
        }
      }
      v_o = v;
      ...
  }
};
```

- Move the indexing outside the if condition
  - Index values now constants after loop unrolling
  - No more replace operations
  - No more indexing logic
- Design issue not flagged by HLS tool
  - would not happen in RTL due to careful coding of mux/demux logic
  - designer thinks software code, causing issues?



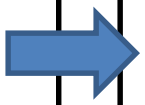
Experienced Designers Stumbling with Sharing

- In a process, a function is called 3 times with different arguments
  - Inside an if-then-else branches
  - Mutually exclusive calls
- Problem: graph inside HLS tool is 3 times bigger then needed
- Heuristics inside HLS tool may not be able to get ideal sharing of
  - one big_op() resources, or
  - multiple small_op() resources
- Recode with one only call site, explicit mux

```
void small_op(…) {
  // custom resource
  …
}

void big_op( … ) {
  …
  for (int i=0; i<10; i++) {
    small_op(…)
  }
  …
}

void proc() {
  …
  if (c1 && !c2) {
    big_op(v1, v2, x, y);
  } else if (!c1 && c2) {
    big_op(v2, v3, x, y);
  } else {
    big_op(v3, v4, x, y);
  }
  …
}
```



```
…
void proc() {
  …
  v_t a1, a2;
  if (c1 && !c2) {
    a1 = v1;
    a2 = v2;
  } else if (!c1 && c2) {
    a1 = v2;
    a2 = v3;
  } else {
    a1 = v3;
    a2 = v4
  }
  big_op(a1, a2, x, y);
  …
}
```

Hardware Designers using HLS

- Hardware designer write code with hardware structure in mind
  - HLS is used to help reduce the amount of details and get to goal faster
- Architecture exploration fast vs. small never happens
  - Great for sale demo, but designer always already have an architecture in mind
- Designer has architecture in mind
  - will often struggle to find a way to code it in SystemC and synthesizable to the desired RTL
  - This gets better with experience, but designer does not have that experience on first project
- Once a design coded well -> huge productivity
- Designer uses abstraction to quickly make improvement to design
  - in ways impossible to do with RTL flow
- Even if code appears to be “lower-level”, there is still immense value!
  - Time scale important for first project (get it in the product), QOR for subsequent projects

Improving SystemC and HLS for Mass Deployment

- To turn RTL designers into HLS designers…
  - The tools must work, in a predictable way
  - Train designers to not expect HLS tool to do anything fancy beyond clean abstraction
  - Clearly document what “software optimization” tool supports…
    … and provide means to check if it did it correctly (i.e. high-quality QOR analysis capabilities)
- Improve SystemC standard such that code can carry intended hardware architecture
  - What should be pipelined,
  - What should be custom resources,
  - What should be unrolled, etc.
- Improve SystemC standard TLM interface for synthesis
  - Resets, parallel non-blocking accesses, decide channel buffering or not
- Improve SystemC datatypes
  - Simulation speed, bit widths known at run-time, complex, etc.

Good QOR needs good designer! HLS will not eliminate hardware designer jobs!

Links

- Accellera SystemC Synthesis Working group:
  - http://www.accellera.org/activities/working-groups/systemc-synthesis
- Accellera SystemC Language Working group:
  - http://www.accellera.org/activities/working-groups/systemc-language
- Please join and participate!

- Numerous commercial HLS tools on the market, and exciting research tools to ckeck out!

Questions? Comments?

Thank you so much for attending!!!